Evaluation of the Performance and Cost of Cloud-Based Robot Motion Planning

Jonathan Lynn
Department of Computer Science
University of North Carolina at Chapel Hill
jonathanlynn@unc.edu

Abstract

We evaluate and analyze the empirical performance and financial costs of using cloud computing to accelerate robot motion planning. In prior work, Ichnowski et al. proposed a method that splits the computation of a robot's motion plan between the robot's own low-powered processor and a cloud-based high-powered compute-optimized server [1]. We implement this method using the compute service offered by Amazon Elastic Compute Cloud (Amazon EC2) on a Fetch robot [2] using 8 degrees of freedom to complete a task in a dynamic environment, as would be required for a consumer-based home-assistance robot. Computing across different data centers around the world, we investigate the network time elapsed during the plan-transmission process, the time taken to complete a task with the robot, and measures of planner performance including the number of vertices and edges generated for each roadmap created during the planning process. Finally, we use these results to estimate the cost model of such pay-per-use cloud services used in this new planning method to evaluate its economic feasibility and desirability compared to a traditional capital expense model.

1. Introduction

Robot motion planning involves the process of moving a robot from a specific initial configuration to a desired goal configuration while avoiding obstacles and self-collision, often optimizing for a cost function such as minimizing the distance travelled or the time to complete the task. The robot searches for a free passageway in the configuration space and adjusts its own configuration by moving and/or rotating its joints in order to successfully navigate to its goal according to the requirements of the task. Because of the many factors and specifications of real-world scenarios as well as the inherent requirements and

methods of planning, robot motion planning can be a computationally-intensive process that actually restricts the nature of some of the problems we can solve. Thus, cloud computing offers an exciting possibility for tackling this issue. Since this directs us in a new approach towards robot motion planning, we will evaluate the potential impact it can have in a real system and a physical scenario.

We use the Fetch robot for our experiments in this paper [2]. The Fetch robot consists of a 7 degree-of-freedom arm with a gripper at the end, a mobile base, a prismatic torso lift, and a rotating RGB-D (RGB camera with depth sensing) head camera. It also contains an onboard 2.9 GHz Intel i5-4570S processor with 4 cores. This is powerful enough to efficiently compute motion plans for simple scenarios, but situations in the real world present motion planning problems that largely exceed the computational power of the Fetch robot's built-in CPU. With the robot's 7 degree-of-freedom mobile arm and prismatic torso lift joint in addition to the need to account for dynamically-moving obstacles in the environment, we have a higher-dimensional planning problem that requires much more computational power for generating solutions and increasing their optimality.

The massive compute power of cloud servers is a great solution to the limitations of the often weaker and power-constrained local processors that robots have. With a connection to the internet, we can access the high-performance systems with many processor cores to better suit the compute needs of more complex motion-planning problems like those present in dynamic real world scenarios. Because the robot's local processors are better suited for local tasks such as actuator control and handling local planning, Ichnowski et al. proposes a method that strategically divides a roadmap-based motion-planning task between the local robot and remote cloud server, accounting for the network transmission requirements in order to compute a motion plan that meets the high demands of a dynamic environment [1].

The roadmap-based robot computation in the method sends requests for a roadmap to reach its goal configuration with information about the environment and time to the server and receives the plan it needs to traverse in order to move from its current configuration to a goal configuration. Incorporating the plan into its roadmap, the robot then checks for changes in the environment and, if required, re-computes or improves the path using Anytime D* [3], considering only the collisions along the path and the dynamic obstacles. Once this step is complete, the robot then sends commands to its controller to move and/or rotate the joints, and the process repeats.

Unlike the robot, the cloud server does not need to detect changes in the environment, and so the roadmap-based cloud computation only needs to consider the static obstacles and self-collision avoidance. The cloud server then generates a probabilistic roadmap (PRM) [4] for the robot to use to find the shortest path between the robot's start and goal configurations while avoiding obstacles. This is done by taking random samples in the robot's configuration space, checking whether these points (including the start and goal points) are collision-free and linking these points with each other to then find the shortest connected collection of collision-free edges.

Specifically, the roadmap on the cloud is computed using the *k*-nearest neighbors PRM* [5] of the vertex that is being considered (the current configuration). This method also enables lock-free parallelization [1] and uses an Incremental Roadmap Spanner (IRS) [6] to reduce the size of its roadmap. It then adjusts its compute-time limit by the duration for the robot to respond to the plan in order to get it to the robot by its target completion time. A compact and relevant subset of the resulting roadmap is then selected via a time-constrained modified Dijkstra method and serialized in binary across the network to the robot for execution using HTTP and TCP/IP.

Ichnowski et al.'s results shed light on the relationship between latency and solve-time/graph quality as well as that between the density of the roadmap sent and the quality of the motion plan. Specifically, in both scenarios, there is a threshold at which cloud computation and dense graphs no longer become the optimal choice, especially when considering the costs of these services. This observation is precisely what leads us to our investigation in this paper.

2. Problem Definition

Our purpose is to evaluate the potential for cloud computing to improve the performance of motion planning based on the method proposed in Ichnowski et al.'s paper using Amazon's compute services. We also want to investigate the costs associated with this approach and compare it to alternatives such as using an on-site machine.

Let C_i be the cost of operating the cloud computation service in a typical home-assistance robot scenario for a data center, i. C is defined as follows:

$$C_i(t, T(l, G(V, E)), q);$$

where t is the usage of the server in years, T is the total time the robot takes to execute the entire plan and reach its goal state, l is the network round-trip time, G is the graph generated by the cloud for the motion plan, V and E are the number of vertices and edges in that graph, respectively, and q encapsulates all other variable costs such as maintenance, energy, depreciation, etc.

The cost model is directly correlated with server usage. Modelling it as an aggregate value, the cumulative cost increases throughout the lifetime of the server. The specific daily usages, however, are dictated by the total plan time T for each task. A longer time requirement for a certain task or chore will lead to a higher daily server usage value and drive up the overall operating cost of server utilization. The total plan time, T, is in turn affected by the network time l and the size of the graph generated for the motion plan G. The network time directly increases the overall time it takes to finish the execution of a plan, but also limits the amount of time for the planning process. This may actually decrease the size of the graphs generated, reducing the complexity and time required to complete the motion. Finally, we introduce the variable q to capture all the remaining

variable costs associated with the operation and maintenance of the compute servers.

In this paper, we will analyze two cost structures, $C_{i,opex}$ and $C_{i,capex}$, which respectively model the cost of utilizing a pay-per-use cloud service (i.e. Amazon EC2) and the cost of purchasing and maintaining a comparable personal server machine over time. We compare these two expense models and find the equilibrating time t^* to make conclusions on the circumstances that contribute to the desirability of each of the two models.

3. Amazon's Compute Services

The Amazon Elastic Compute Cloud (Amazon EC2) offers secure and on-demand cloud-compute capacity with a simple management interface and configuration capabilities, allowing users complete control over the resources needed for computation. Amazon EC2 is also optimized for launching and booting new server instances, allowing production to be scaled at minimal costs to usage time.

The cloud service used in our experiments come from the 4th generation compute-optimized C4 instances. These instances offer the lowest price over compute-performance among all EC2 instances. The c4.8xlarge instances possess the 2.9 GHz Intel Xeon E5-2666 v3 (Haswell) processor with Turbo Boost technology. It is virtualized with 36 vCPUs and contains 60 gigabytes of memory. The c4.4xlarge instances are identical to the c4.8xlarge instances except that they have 16 vCPUs and 30 gigabytes of memory [7].

Amazon's specification of vCPUs (virtual CPU) is not completely analogous to the number of cores its instances have. Since these instances are virtualized and because both the c4.8xlarge and c4.4xlarge instances have multithreading enabled [7], each vCPU represents a single thread on a multithreaded core. Thus, every 2 vCPUs corresponds to 1 core [8]. That gives us an 8-core equivalence for the 16-vCPU c4.4xlarge instances and an 18-core equivalence for the 36-vCPU c4.8xlarge instances.

We ran experiments using 6 different Amazon data centers: Northern Virginia, Ohio, Northern California, Oregon, Ireland, and Mumbai, each geographically further away from our robot location in Chapel Hill, NC. The prices for the two instance types we used across these 6 regions are shown in Table 1.

	Region Name	c4.8xlarge	c4.4xlarge
1	US East (N. Virginia)	$1.591	$0.796
2	US East (Ohio)	$1.591	$0.796

3	US West (N. California)	$1.993	$0.997
4	US West (Oregon)	$1.591	$0.796
5	EU (Ireland)	$1.811	$0.905
6	Asia Pacific (Mumbai)	$1.756	$0.878

Table 1: On-Demand Cost of Instances Across Regions (as of Apr. 28, 2017) [9]

4. Methods and Results

The scenario used in Ichnowski et al.'s experiments introduces an 8 degree-of-freedom task in which the Fetch robot's prismatic torso lift joint and arm are mobilized for a pre-grasp task that involves the arm moving from an initial configuration behind a table to a pre-grasp configuration above it at the Coke bottle (Figure 1). During the planning and execution process, one of the researchers waved a poster-tube in the trajectory of the roadmap to interfere with the execution of the roadmap. The obstacle was tracked by pinpointing two colored strips attached to the poster-tube using the RGB-D information received from the point-cloud feed of the head camera and calculating the transformations between the midpoints of these two strips. Figure 1 shows the full process of the Fetch-Grab task. As aforementioned, the job begins at an initial configuration (Top-left of Figure 1). The Fetch robot then plans a roadmap given the initial detected position of the dynamic obstacle. As the obstacle moves, the robot rapidly re-plans its robot to avoid collision with the obstacle until it reaches its goal configuration shown at the bottom-right of Figure 1.

Using the same scenario from this paper, we execute the proposed algorithm on the Fetch robot using the c4.4xlarge and c4.8xlarge instances across the six different regions as the cloud-compute server instead of the Renaissance Computing Institute's (RENCI) 32-core system (four Intel x7550 2.0-GHz 8-core NehalemEX processors) as in Ichnowski et al.'s experiments, which is located approximately 6 km away and connected by a high-bandwidth, low-latency network.

In order to familiarize ourselves with the AWS console and its usage, we first created and launched a free-tier test instance (t2.micro instance type). We launched the instance with an Ubuntu Server image and configured it to accept TCP traffic to and from our robot at a predefined port. Using the AWS Command Line Interface, we then set up an automated script that allowed us to start and stop the instance and, using SSH and SCP protocol, connect to and send our server software over to it without using the console.

After configuring the t2.micro instance, we connected to it via the command line and launched the cloud-planning software used in Ichnowski et al.'s method. We then supplied the IP address of the t2.micro instance to the

planning software on the Fetch robot before starting our trials.

In order to hold a consistent benchmark across our experiments, we simulated 50 continuous runs with the motors of the Fetch robot disabled and replaced the poster-tube with a virtualized baton swinging in uniform horizontal motion. However, it must be noted that despite of the uniform swinging motion, the robot did not predict the location of this obstacle but detected it throughout the task. We also set an upper-bound timeout limit of 120s for each run.

5.1 Cloud Performance on 36-vCPU Instances

For the first set of experiments, we set up and launched six identical c4.8xlarge instances across six different AWS regions (one instance per region) and configured the Fetch robot to communicate with each via its unique IP address. This was done by repeating the process outlined above six times, each time specifying a different AWS region to work in. The six regions included the four regional data centers in the United States: N. Virginia, Oregon, N. California, and Ohio, and two overseas: Ireland and Mumbai.

We ran a trial of 50 runs for each of the six regions on the Fetch robot, each time logging the start and end times of the task of reaching the goal pre-grasp configuration, the total network time for each roadmap transmission, and the number of edges and vertices of the local graph on the robot. From these logs, we calculated the total time elapsed for the Fetch robot to complete its task and found the average network round-trip time across all the roadmap transmissions for each of the 50 runs. The results, along with the distances of each data center from Chapel Hill, are shown in figures 2, 3, 4, and table 2.

From these initial results, we find that the physical distance of the data center to the client location (Chapel Hill, NC) is generally proportional to the time-elapsed for the Fetch robot to reach the goal configuration and inversely-related to the number of edges and vertices generated in the roadmap graphs. It is also directly proportional to the network round-trip time. However, there is a discrepancy in this trend for the goal-time between the N. Virginia and Ohio data centers. Specifically, the goal-time elapsed for the compute server run in N. Virginia had a higher median but was accompanied with a denser graph. The denser graph offers a possible explanation that, because of a shorter network round-trip time between Chapel Hill and the N. Virginia data center, more time was allotted to roadmap-generation and so, as a result, the robot spent more time executing the denser roadmap supplied by the instance in N. Virginia.

With all these results in mind, one thing we must take note of is that the physical distances between Chapel Hill and each of the data centers are not necessarily indicative of all of the factors that influence the transmission time such as the underlying network topology, bandwidth, whether the routes are direct, and the size of the graphs transmitted. Thus, although our results do implicate that there is a correlation between the physical distance between the client and data center and the network round-trip time, we cannot make assumptions about network performance solely based on the physical distances

between the data center and client location, which is an important consideration when implementing a cloud-based service.

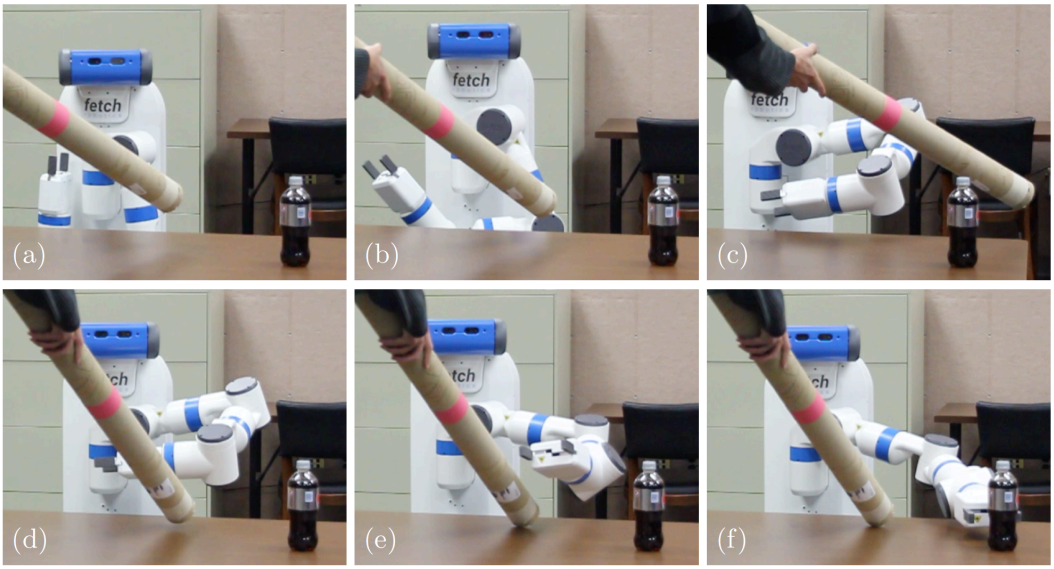

Figure 1: Fetch-Grab Scenario (From [1])

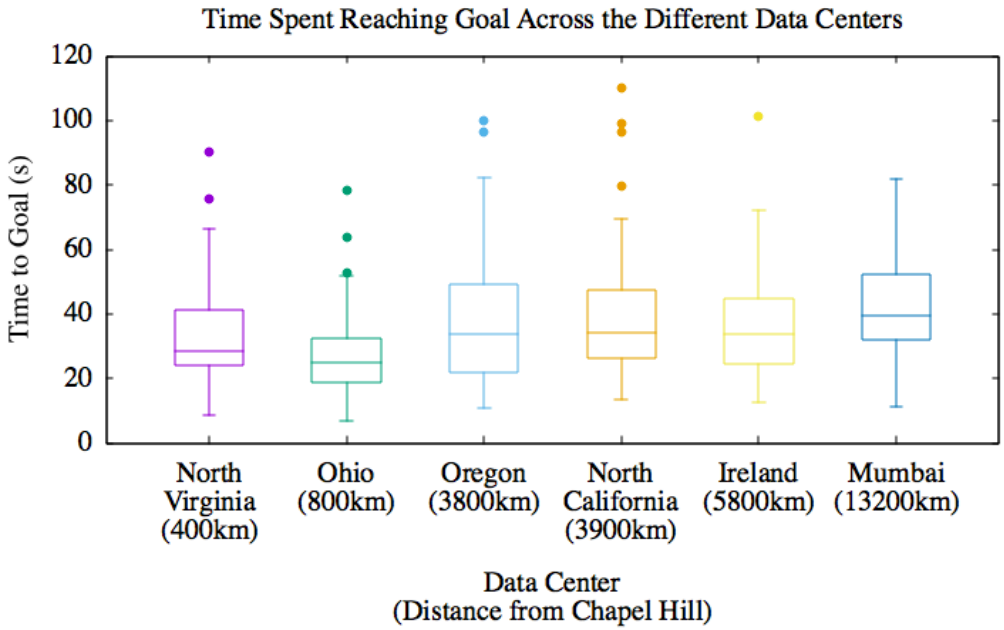

Figure 2: Goal Time Across Data Centers Running 36-vCPU Instances

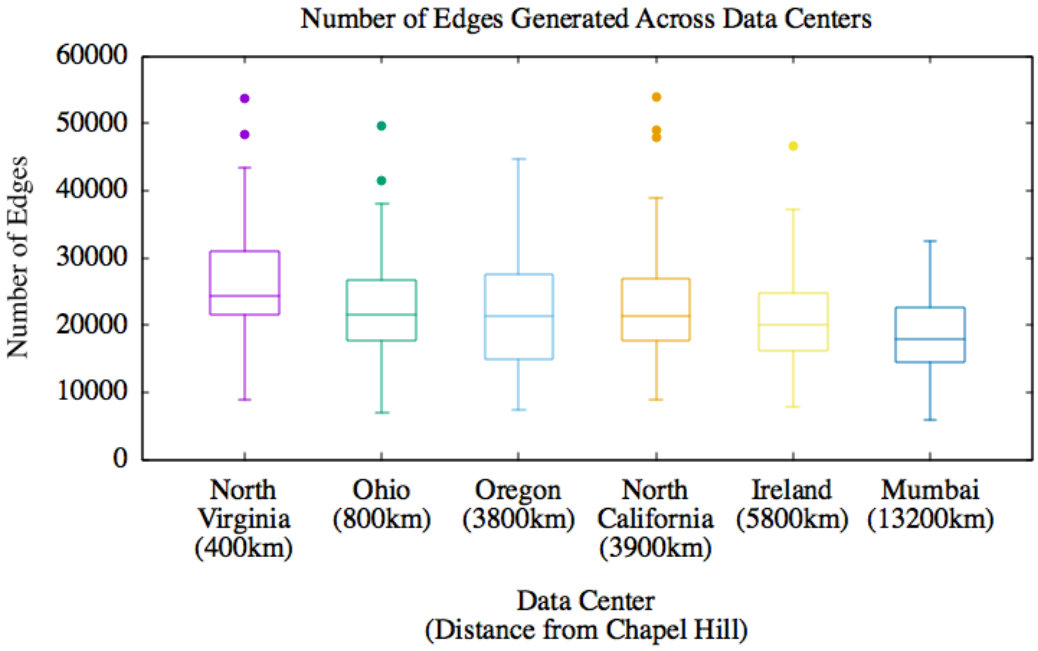

Figure 3: Edges Generated Across Data Centers Running 36-vCPU Instances

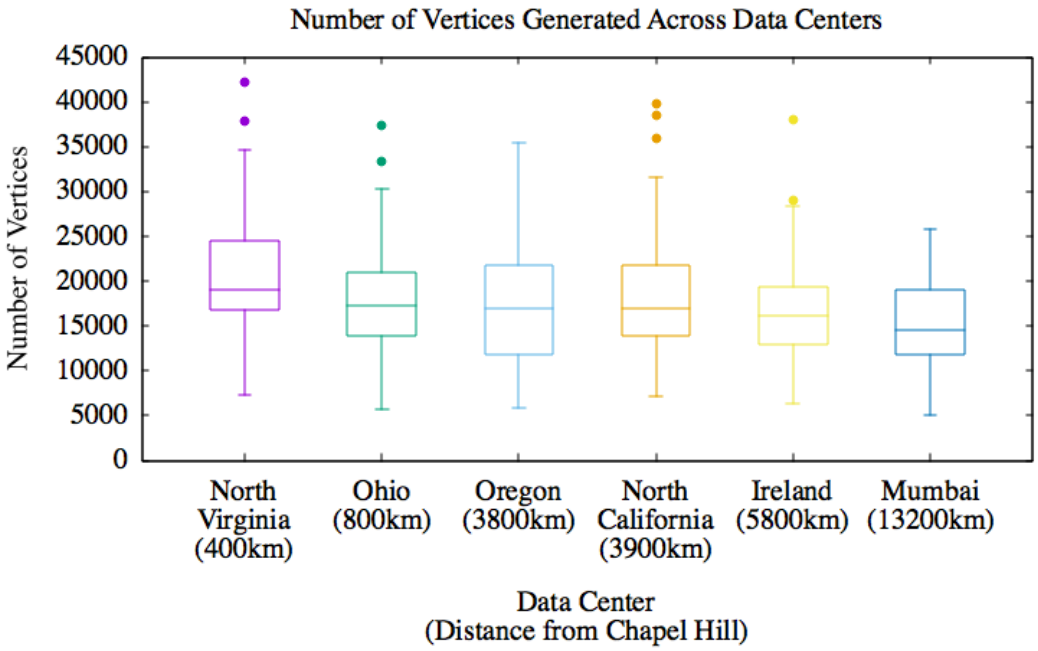

Figure 4: Vertices Generated Across Data Centers Running 36-vCPU Instances

Region Name	36-vCPU Network Round-Trip Time (ms)
US East (N. Virginia)	79.1
US East (Ohio)	90.9
US West (N. California)	177.8
US West (Oregon)	197.5
EU (Ireland)	205.3
Asia Pacific (Mumbai)	347.4

Table 2: Average Network Round-Trip Time Measured Across Regions on 36-vCPU Instances

5.2 Cloud Performance on 16-vCPU Instances

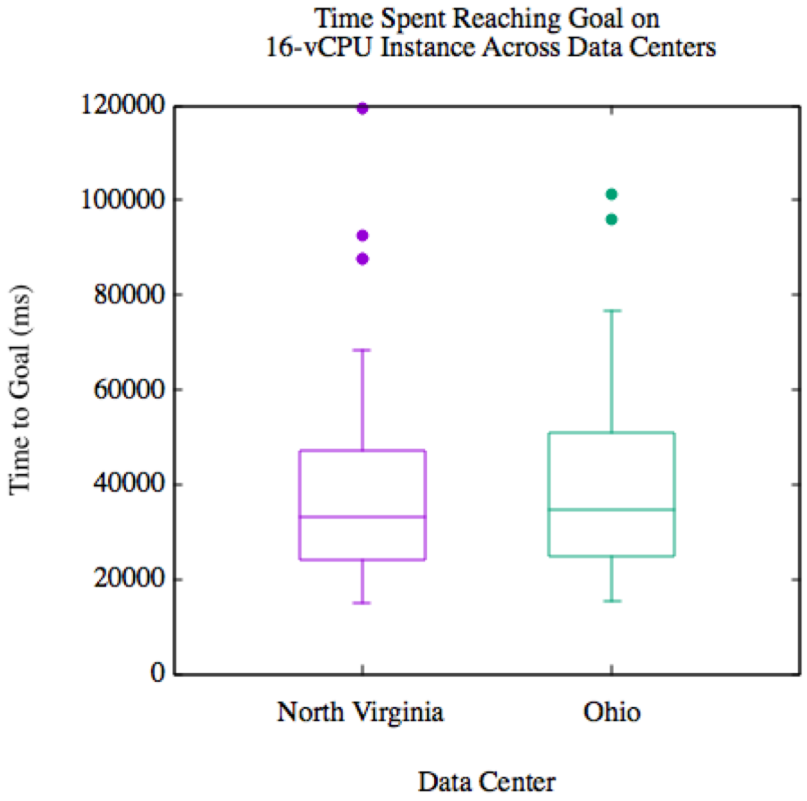

Figure 5: Goal Time Across Regions for 16-vCPU Instances

In order to investigate the effects of reducing the cloud-service costs on the performance of the cloud-based motion planning method, we ran two additional trials of 50-runs with the same scenario on the N. Virginia and Ohio data centers, downgrading the performance of the server to the 16-vCPU EC2 c4.4xlarge instance. The procedures for the instance configuration, trial execution, and analysis were the same as those for the c4.8xlarge instances. The results are shown in figures 5, 6, and 7.

Under this adjustment, the time-to-goal becomes shorter for the N. Virginia data center, consistent with our previous assumptions. In addition, overall, the c4.8xlarge (36-vCPU) instance decreased the goal time by 26.42% and 38.06% for the N. Virginia and Ohio data centers, respectively.

However, the network transfer times and graph sizes did not reflect a consistent narrative. The results show that the increase in quality of the roadmaps generated from the higher-powered instance (36-vCPU) was not universal, with

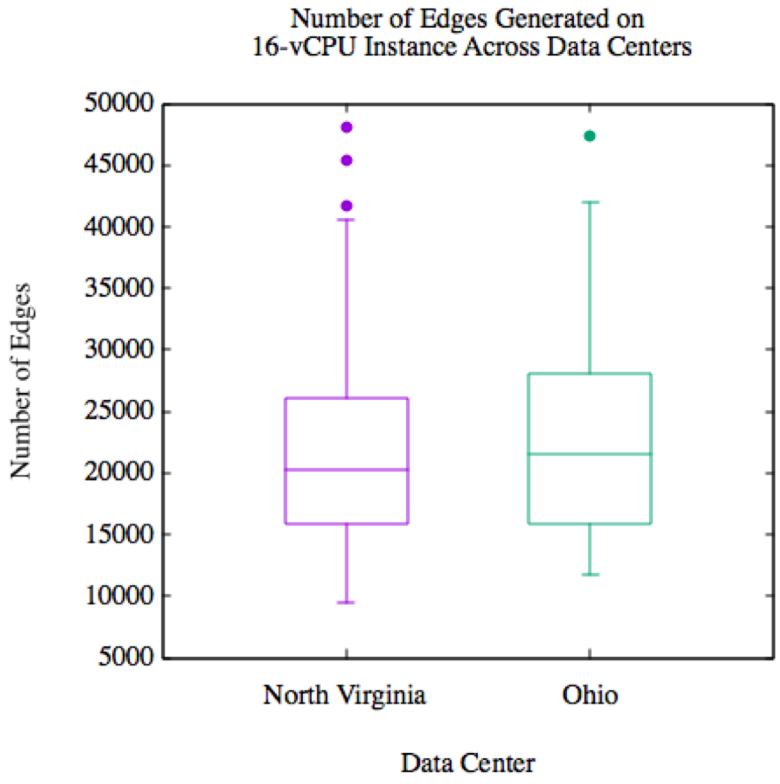

Figure 6: Number of Edges Generated Across Regions for 16-vCPU Instances

Ohio showing no significant optimization. N. Virginia saw an 18.46% increase in the number of vertices and a 16.47% increase in the number of edges, while Ohio only saw a 3.36% and 0.16% increase in the number of vertices and edges, respectively. Moreover, there was a higher share of total time-to-goal allocated to the network at the N. Virginia center, which explains the corresponding smaller

number of edges and vertices generated in the roadmap. While the network time remained the same within error for the N. Virginia data center, the network time decreased from 90.9ms to 56.2ms for the Ohio data center.

We evaluated the cost differentials between the c4.8xlarge and c4.4xlarge instances across the N. Virginia and Ohio data centers and found that it actually cost 47.06% and 23.80% more than the lower-tier 16-vCPU instance for the two centers respectively.

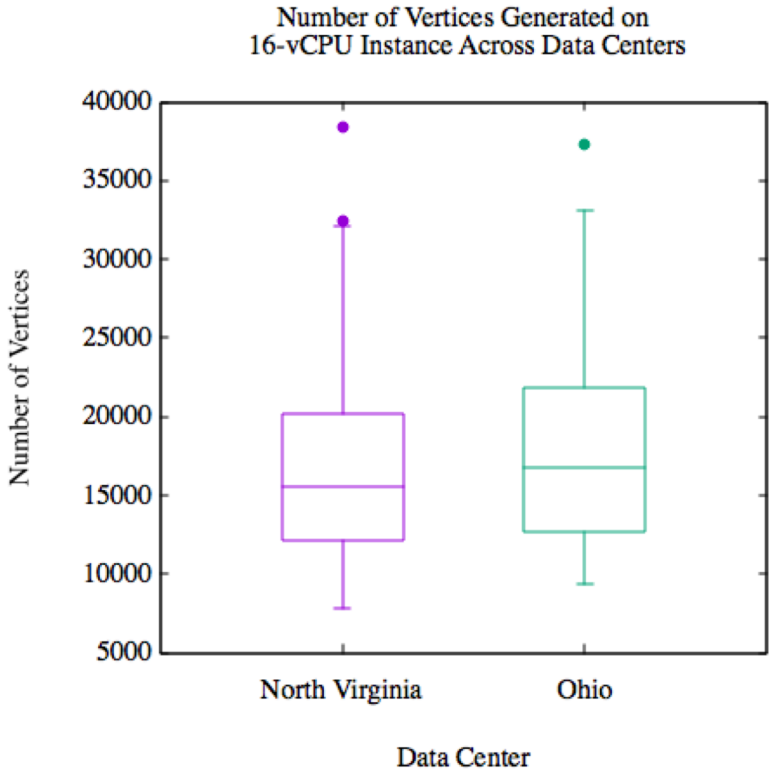

Figure 7: Number of Vertices Generated Across Regions for 16-vCPU Instances

5.3 Evaluating the Economic Models

The final evaluation we make in this paper is the cost breakdown of using a cloud-based compute service like AWS versus the purchasing of a traditional high-powered in-house machine like the RENCI 32-core system. We compare the economic desirability (profitability) as well as time spent on maintenance as a function of time for both the service model utilized by Amazon EC2 and the capital expense model of the traditional method and find the equilibrating time at which both methods are equally desirable.

According to the US Department of Labor Bureau of Labor Statistics, the

average time spent per day in a traditional 2-person household on household activities excluding administrative tasks totals 198 minutes [10]. Using this as an analogous benchmark to estimate the amount of time a typical home-assistance robot will spend on household tasks and, correspondingly, the amount of time the cloud server will need to run, at the current rate for EC2 c4.8xlarge at the N. Virginia region of $1.591/hour [9], we calculate a $5.30 daily cost for using the Amazon cloud service. We call this the operating-expense (opex) model.

We call the alternative method of purchasing an in-house server the capital-expense (capex) model. Configuring a machine of comparable processor benchmarks as the Amazon EC2 c4.8xlarge computer service costs a fixed price of $10,981, requiring two Intel Xeon e5 2660 v3 processors. Such a processor also generates 105W of heat. With energy costs of 11 cents/kWh with Duke Energy [11], the costs of running the machine amount to $0.2772 per day. Using

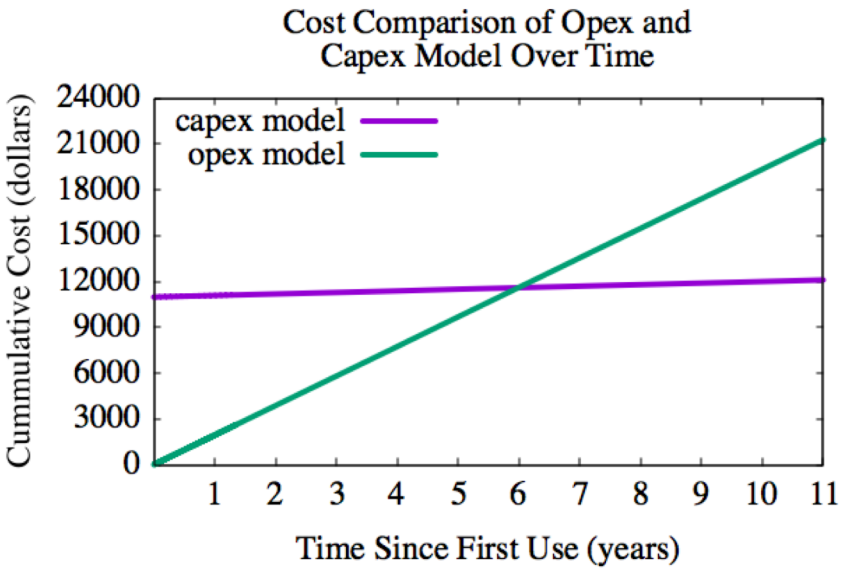

Figure 8: Cost Comparison Over Time Between Pay-per-use Server Subscription Model and Traditional Comparable Machine-Purchase Model

this value, we calculate that the capex model, adjusting for the costs of energy over time, is as shown in Figure 8. From these calculations, we find that the equilibrium point between the opex and capex models occurs at t=6.00 years at a cost of $11,600. Beyond this point, the traditional capex model is actually more favorable than our proposed opex model.

One thing to keep in mind is that the performances of the network connection to the cloud between the two models differ. The locally installed server has ultra-low latency and high bandwidth as compared to the network roundtrip time of 79.1ms for a connection to the 36-vCPU instance at the N. Virginia data center. In practice, depending on the specific requirements of the

robot task and when its duration is much longer, this difference may be reflected in an additional cost (whether direct or indirect) that can potentially tip the favorability over to the capex model.

5. Conclusion

An empirical evaluation of the performance of the cloud-based motion planning methods validate our initial assumptions that as the geographical distance from the compute server increases, the total time to reach the goal configuration whereas the quality of the roadmaps generated decreases, which is due in part to the increased network latency and potential for network congestion resulting from the increased distance. However, when the generated graphs become too dense, we get a sacrifice in the network transfer time as well as the total time taken to reach the goal. Therefore, when working with closer data centers, higher-powered servers may not necessarily always be preferable over a slightly lower one.

This leads us to our analysis on the financial cost side. As it turns out, the economic benefit of using a pay-per-use cloud service is certainly substantial over the purchase of a comparable machine when considered in the short-run. However, when the usage time gradually increases, the benefits diminish and eventually force this model to be less preferable to the traditional one. Moreover, if we take away our assumptions of uniform latency times, the saved latency as a result of the closer location of an in-house server may in fact allow for a lower-powered machine that can reduce the overhead costs for the traditional model, and may in fact give us a much earlier equilibrium time.

6. Future Work

The results of our experiments presented many opportunities for future research and investigation. First, the discrepancy in the comparative results between the Ohio and N. Virginia data centers justify additional experiments that control for other run-time conditions such as data center-utilization and network traffic. One method can be to have the trials run at the same time-of-day and day-of-week across all centers.

Another opportunity for further evaluation is making a similar comparison as performed in our experiments but instead, hold the location of the server constant and making the client/robot location variable. This can be simulated by choosing one region's data center as the computer server and using a mobile machine that has comparable benchmarks as the Fetch robot and executing the trials in different geographic locations (i.e.: in a coffee shop, library, on-campus lab, off-campus residence, etc) to investigate the effects of different network qualities and the robot-server dynamic when the robot moves to different places.

Finally, the limitations in our economic model revealed many alternative routes of investigation that could strengthen the extendibility and accuracy of such a tool. Instead of focusing on the comparisons of the current pay-per-use cloud service model against purchasing and using a comparable machine at home, there is in fact much more value in generalizing these comparisons to evaluate the cost benefits of simply having cloud optimizations or what level of optimization is needed in motion planning. These comparisons will require quantifying the costs of lost computation time and gained roadmap quality, but also need to take into account costs of maintenance, machine depreciation, software upgrades, comparison of network time, and various other factors that can play a significant role especially if we consider these models over time. Instead of focusing on an equilibrium point and looking at the cumulative costs as we did in this paper, there may be more value in quantifying the unit financial costs of a certain robot-cloud suite over usage time that specifies the compute server benchmarks, motion-planning needs, and other application-specific details. Or, on the other hand, we can even look into predicting the cheapest planning option given the required performances, or find the best performance for a given cost-level.

Acknowledgments

The research could not have been completed without the guidance and collaboration of Dr. Ron Alterovitz, Jeffrey Ichnowski, and Dr. Gary Bishop.

References

1. Jeffrey Ichnowski, Jan Prins, and Ron Alterovitz, "Cloud-based Motion Plan Computation for Power-Constrained Robots," in *Algorithmic Foundations of Robotics (WAFR 2016)*, Dec. 2016.
2. Melonee Wise, Michael Ferguson, Derek King, Eric Diehr and David Dymesich, “Fetch & Freight: Standard Platforms for Service Robot Applications,” in Workshop on Autonomous Mobile Service Robots, held at the 2016 International Joint Conference on Artificial Intelligence, NYC, July 2016
3. Likhachev, M., Ferguson, D.I., Gordon, G.J., Stentz, A., Thrun, S.: Anytime dynamic A*: An anytime, replanning algorithm. In: ICAPS. (2005) 262–271
4. Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.: Probabilistic roadmaps for path planning in high dimensional conguration spaces. IEEE Trans. Robotics and Automation 12(4) (1996) 566–580
5. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. Int. J. Robotics Research 30(7) (June 2011) 846–894
6. Marble, J.D., Bekris, K.E.: Asymptotically near-optimal planning with probabilistic roadmap spanners. IEEE Transactions on Robotics 29(2) (2013)

432–444
7. Amazon: EC2 Instance Types. https://aws.amazon.com/ec2/instance-types
8. Tan, Chin Khoon, “Demystifying the Number of vCPUs for Optimal Workload Performance”, Amazon Web Services, Aug. 2016.
9. Amazon: EC2 Instance On-Demand Pricing. https://aws.amazon.com/ec2/pricing/on-demand/ Accessed Apr. 2017.
10. Bureau of Labor Statistics: Household Activities. https://www.bls.gov/TUS/CHARTS/HOUSEHOLD.HTM
11. Duke Energy: Duke Energy Progress rates in North Carolina to decrease Dec. 1 due to lower fuel costs. https://news.duke-energy.com/releases/duke-energy-progress-rates-in-north-carolina-to-decrease-dec-1-due-to-lower-fuel-costs